

Analyzing biosignals using the R freeware (open source) tool

Frank Borg *

January 15, 2013

Abstract

For researchers in electromyography (EMG), and similar biosignals, signal processing is naturally an essential topic. There are a number of excellent tools available. To these one may add the freely available open source statistical software package R, which is in fact also a programming language. It is becoming one of the standard tools for scientists to visualize and process data. A large number of additional packages are continually contributed by an active community. The purpose of this paper is to alert biomechanics researchers to the usefulness of this versatile tool. We discuss a set of basic signal processing methods and their realizations with R which are provided in the supplementary material. The data used in the examples are EMG and force plate data acquired during a quiet standing test.

1 Introduction

In biomechanics the application of electromyography is especially a field rife with various signal processing methods. One reason for this is the complexity of the EMG signal which requires processing if one wants to proceed beyond the simple level of on-off interpretation. A number of standard processing tools can be found in EMG softwares by commercial vendors. Researchers are usually also interested in new, or modified processing methods, which may take long to implement in commercial softwares if it happens at all. In this case one can choose "prototyping" tools like LABVIEW, MATHCAD, MATLAB, OCTAVE, SCI LAB, and so on. Over the the years we have for instance frequently

*Kokkola University Consortium Chydenius, Finland. Email: borgbros@netti.fi

used LABWINDOWS CVI (a C-based graphical programming tool) and MATH-CAD for data acquisition, visualization and analysis. One common problem in data intensive projects is that the data and analysis documents get scattered among computers and data disks. Our solution is to transfer all data to a single database (in our case based on MySQL). Subsets of data can then be selected by sql searches and exported as csv-tables or similar. The salient point is that all the analyses can be gathered in one single script file using the R-tool (R Development Core Team, 2010). Running this script generates all the graphs, figures and statistical reports that one may need for the project. This script is easily modified when needed and can be shared among collaborators and other researchers. We believe that R provides a useful platform for defining and using various signal analyzing algorithms in EMG studies, which may stimulate further refinements and developments through collaborative efforts. As noted by (Everitt and Hothorn, 2010, Preface):

(...) R has started to become the main computing engine for statistical research (...) For reproducible piece of research, the original observations, all data processing steps, the statistical analysis as well as the scientific report form a unit and all need to be available for inspection, reproduction and modification by the readers.

With this paper and the supplementary material we hope to give a demonstration of the potential uses of R in biosignal processing.

2 What is R?

R is an interpreted functional programming language for statistical computing and data visualization created by Ross Ihaka and Robert Gentleman (University of Auckland, New Zealand). R is part of the GNU project and since 1997 it is developed by the *R Development Core Team*. The version R 1.0.0 was released on 29 February 2000. The official project website is www.r-project.org where one can naturally download the software (binaries, or source code) and manuals (Venable et al, 2009). R is an implementation of the S language developed at the AT&T Bell Laboratories around 1975 (by Rick Becker, John Chambers and Allan Wilks), and is influenced by the LISP dialect SCHEME created at the MIT AI Lab, also around 1975 (by Guy L Steele and Gerald Jay Sussman). A large part of the software is written in the R language itself, but core functions are written in C and FORTRAN. As an interpreted language R may be slower than say pure C-programs. Klemens, who advocates C as general scientific programming tool, gives an example with running the Fisher test 5 million times: the speed relation came out as about 1:30 in favour of C (Klemens, 2009).

Thus in speed-critical cases, such as where we have large simulation samples, it may be necessary to revert to compiled C programs. However, in many other cases the versatility of R will outweigh the possible gains from optimizations for speed. The R tool is continually extended by packages contributed by an active community; there are presently more than 2000 packages available, see <http://cran.r-project.org/web/packages/>.

Downloading and installing the R software on your computer should require nothing more than basic computer skills. A useful companion is the R scripting tool NPPTOR (<http://sourceforge.net/projects/npptor/>) which is an add-on to the editing tool NOTEPAD++ which can be downloaded from <http://notepad-plus.sourceforge.net/uk/site.htm>. Since documentation about R is easily found on the web we will move on to describe what we can do with the software.

3 Importing data

The first thing we want to do is to get our data into the workspace. R can access various databases directly but the most typical situation is where we have, say, EMG data exported on a text format. We use R as console program, so the command for reading a simple ASCII table of data from a file into variable `M` using the `read.table()` function is:

```
M <- read.table("C:/.../myData.asc", header = FALSE)
```

where the first argument contains the path to your data file. If the columns are separated for instance by ";" (instead of space) we have to add to the arguments `sep = ";"`. Note also that the assignment operator in R is an arrow `<-` and not `=`. We have assumed that the data table had no header. In this case column names can added by the command,

```
names(M) <- c("colName1", ..., "colNameN")
```

if there are N columns. You can then refer, for instance, to the first column as `M$colName1`. The other way to extract the first column is as `M[, 1]`. One can inspect the content of the table read into `M` by the command `edit(M)`. Finally you can quit R by the command `q()`. Instead of typing and running the commands at the console in a sequence, you can collect them in a script file (an

ordinary text file) with the extension R, named say `myScript.R`, and run it by the command `source("C:/.../myScript.R")` where the argument contains the path to your script file.

4 Analysing data

4.1 Plotting data

The next thing you want to do is to inspect your data. For a quick plot of the data in the example above use the command,

```
plot(M$colName1, type = "l")
lines(M$colName2, type = "l")
```

The second line adds the graph of the second column to the same plot. The `plot` function has various arguments for controlling colours, titles, scales, and so on. Information about the function `plot` is obtained by the command `?plot`. An interesting feature is that with a few lines of code it is for example possible to plot histograms/graphs of hundreds of variables and print them to a pdf document for quick browsing on the computer.

After these preliminaries we can move on to the processing of data. In this connection we must explain how we define new functions in R.

4.2 Basic signal processing – some examples

4.2.1 User defined functions

In most programming tasks it is convenient to be able to define your own functions. We give a simple example how it works in R. Let us say that we want to sum the elements of vectors using a function `sumVec(V)` which returns the sum and mean of a numeric vector `V`. This can be defined in R by:

```
sumVec <- function(V, start = 1){
  n <- length(V)
  sm <- 0
  for(i in start:n){
    sm <- sm + V[i]
  }
  mn <- sm/(n - start + 1)
  return(list(sum = sm, mean = mn))
}
```

We can call this as `result <- sumVec(V)` which puts the sum and mean of the elements of the vector `V` into to a variable `result`. Here we have also demonstrated the very useful `list` structure in R. The sum (`sm`) and mean (`mn`) are put into a list with respective (arbitrary) names `sum` and `mean`. The variables are then referenced as `result$sum` and `result$mean` after the call

```
result <- sumVec(V).
```

It is an advantage to have a function return a `list` since it is then easy to modify the function by adding new variables to the output `list` without affecting previous uses of the function. The function example also illustrates an other aspect of R function; that is, we may have arguments with default values, like `start` as in the example. If the argument is not listed, as in

```
result <- sumVec(V),
```

it will use the default value (`start = 1`). In many functions we have set as default `plot = FALSE` for a variable `plot`, which means that the results will not plotted unless one adds `plot = TRUE` to the arguments.

Functions can also have other functions as arguments, as for example in the case of `EMG_spec` below. This example also illustrates the similarity with the common C and JAVA syntax. User defined functions can be collected in a separate file `myFunctions.R` which can be made active (loaded into the workspace) by calling it using

```
source("C:/.../myFunctions.R")
```

This corresponds to the `include` statement of header files and source files in C programming. The EMG processing functions to be described below are collected in the file `EMGfuncs.R` in the supplementary material to this paper. The hash-symbol `#` is used for comment lines in the R scripts.

4.2.2 Simulated data

Is useful to have access to various test data, and as example we have implemented a standard algorithm (Hermens et al, 1999, pp.70-71) in `EMG_sim` which generates simulated EMG data of desired length. This function returns a `list` where the data is contained in the component `sim`. For instance one can probe the spectrum function using the test data as follows:

```
mysim <- EMG_sim(3000)
myspec <- EMG_spec(mysim$sim, plot = TRUE)
```

4.2.3 Rectification, RMS and turns

Rectifying raw data means simply taking the absolute value of the elements. For EMG data it may be useful to first subtract any offset (bias) from the raw data. Thus the rectification of V could be written as $\text{abs}(V - \text{mean}(V))$ and the `EMG_rect` function in `EMGfuns.R` becomes very simple. It is noteworthy that effect of the rectification of EMG is similar to the rectification of AM radio waves whose purpose is to enhance the low frequency components which encode the voice signals. For EMG the low frequency "voice" part corresponds to the encoded force (Borg et al, 2007).

A bit less trivial from the programming point of view is the `EMG_rms` function which computes the Random Mean Square of the EMG and can basically be represented as

$$\sqrt{\frac{1}{\Delta T} \int_{t-\Delta T/2}^{t+\Delta T/2} EMG(u)^2 du}. \quad (1)$$

To write these sorts of filtering or enveloping functions it is convenient to use the built-in `filter(V, filtc, ...)` function. This takes input data V and outputs a vector with elements

$$y[i] = \text{filtc}[1] \cdot V[i + o] + \dots + \text{filtc}[p] \cdot V[i + o - (p - 1)], \quad (2)$$

where `filtc[i]` are the filter coefficients. The offset o depends on the argument sides such that `sides = 2` corresponds to zero lag with $o = (p - 1)/2$ if p is odd. For the moving average one uses `method = "convolution"`. The function `EMG_rms` has an argument `DT` which determines the size in milliseconds of the moving window over which one calculates the RMS. This function also illustrates a special feature of R: the use of the `'...'` argument.

```
EMG_rms <- function (V, sampFreq = 1000, DT = 250, plot = FALSE, ...)
{ #part of the function declaration
  rectV <- sqrt(filter(rectV^2, filter1, sides = 2,
    method = "convolution", ...))
#rest of the function declaration
}
```

In this case it means that we can pass arguments to the `filter` function employed by `EMG_rms`. For instance, `filter` has an argument called `circular`, and via `EMG_rms(..., circular = TRUE)`, we can pass a value (TRUE in this case) to this argument of the function `filter`.

A version of the classical method of counting turns (Willison, 1963, 1964) is implemented by the function `EMG_turns` and it also uses a moving window `DT` over which one sums the number of turns. The implementation `EMG_wturns`

is maybe closer to the original idea by Willison but the practical difference between the two seems small. The turns functions return a structure

```
list(turns.ps = turns_per_sec, turns.where = turns).
```

The variable `turns.where` contains the time indexes where the turns are counted. (This is also shown in the plot of the function.) This data may be of interest when, for instance, one wants to calculate an entropy metric for the signal.

4.2.4 Time-frequency domain

Part of the inspection of the EMG signal is to study its frequency properties. This is typically performed by calculating the power spectrum of the data. For this purpose one subdivides the original times series into blocks of some time length ΔT , then calculates the power spectrum for these and take their mean as the final power spectrum. For the subdivision one normally use a 50 % overlap which further suppresses the variance of the final spectrum estimate (Press et al, 2002). This method is implemented in the function `EMG_spec`. It uses a default windowing of the data by the filter `filtWelch`. The window size ΔT is given by the argument `DT` in milliseconds. The nominal frequency resolution is then given by $\Delta f = 1/\Delta T$. The function outputs a `list` with the power density estimate in `psd`, which also contains mean (MNF) and median frequency (MDF) in `meanf` and `medianf`. The time-frequency methods naturally rely on the *Fast Fourier Transformation* (FFT) which in R is called by `fft`. Using the *Short Time Fourier Transformation* (STFT), which applies the FFT to subintervals of the time series, we can, for instance, calculate how the mean frequency varies with time. This is implemented by the function `EMG_stft_f`. One use of mean/median frequency is for the study of muscle fatigue as a function of time, which is often associated with a decrease in mean frequency (Lindström et al, 1977).

Using the `fft` transform we can filter EMG signals by suppressing the higher frequency components. The method employed in (Borg et al, 2007) is here implemented by the function `EMG_bw0`. It first calculates the Average Rectified Value, then applies `fft` which gives the Fourier coefficients $c(f_k)$. These are multiplied by a filter factor,

$$c(f_k) \mapsto \tilde{c}(f_k) = \frac{c(f_k)}{1 + \left(\frac{f_k}{f_c}\right)^n},$$

where f_c is the low pass cut-off frequency of the filter and n is the order of the filter. Finally we obtain the filtered signal by applying the inverse `fft` to

$\tilde{c}(f_k)$. This is basically a zero-lag version of the Butterworth filter. With $n = 4$ and $f_c = 1$ Hz we obtained quite a good correspondence between gastrocnemius EMG and the muscle force as expressed by the anterior-posterior COP (center of pressure) during quiet standing. We have also implemented the filter corresponding to a second order critically damped system,

$$c(f_k) \mapsto \tilde{c}(f_k) = \frac{c(f_k)}{\left(1 + i \frac{f_k}{f_c}\right)^2},$$

which is one of the basic models for the EMG-to-force transfer function (Soechting and Roberts, 1975). Note that the function `EMG_crit2` too rectifies the EMG before filtering.

For comparison of EMG vs EMG, or (filtered) EMG vs force etc, the correlation methods are essential. Given two time series x and y we may define a correlation function $c_{xy}(t)$ by,

$$c_{xy}(t) = \frac{\int_0^T \tilde{x}(u) \tilde{y}(u+t) du}{\sqrt{\int_0^T \tilde{x}(u)^2 du} \sqrt{\int_0^T \tilde{y}(u)^2 du}},$$

where \tilde{x} is x with the mean value \bar{x} subtracted, $\tilde{x}(t) = x(t) - \bar{x}$, etc. In the discrete version this is implemented by `EMG_corr` employing `fft` methods. In the frequency domain a coherence function is defined by,

$$\text{coh}_{xy}(f) = \frac{\langle \hat{x}^*(f) \hat{y}(f) \rangle}{\sqrt{\langle |\hat{x}(f)|^2 \rangle} \sqrt{\langle |\hat{y}(f)|^2 \rangle}},$$

where $\langle \dots \rangle$ denotes statistical averaging. This is estimated by `EMG_coh` by dividing the time series into time slices of size `DT` and calculating the Fourier coefficients for these slices, and finally take the averages over the blocks. `EMG_corr` and `EMG_coh` can be used to investigate time lags and phase shifts between signals. In the case that we have a linear relationship, $y = H \star x + n$, with a transfer function H (and uncorrelated "noise" n), we would get

$$\text{coh}_{xy}(f) = \frac{\hat{H}(f)}{|\hat{H}(f)|} = e^{i\phi(f)},$$

where $\phi(f)$ is the phase function of the transfer function. A related time shift τ can then be obtained from $2\pi\tau = d\phi(f)/df$.

4.2.5 Frequency band analysis

As is well known from musical transcription, it is convenient to study sound by how the power (intensity) is distributed over the frequency bands (pitch) as

a function of time. This is useful also in basic signal analysis. The idea is to decompose signals using filter banks. Wavelets can be considered as a special realization of the idea of filter banks (Vetterli and Kovačević, 1995). An interesting hybrid method for "intensity analysis" of EMG has been proposed by (von Tscharner, 2000). The idea is to divide the frequency band of interest, say one from 10 Hz to 200 Hz, into subbands centered on frequencies $f_c^{(j)}$ ($j = 1, \dots, J$) such that the relative bandwidth $BW = \Delta f(j)/f_c^{(j)}$ scales as $1/\sqrt{f_c^{(j)}}$ over the frequency band. Here $\Delta f(j)$ is the frequency resolution of the "mother wavelet" ψ at the center frequency $f_c^{(j)}$. In this way one can provide a distribution of signal power among the frequency bands. The power in the frequency band j at time t is given by $|c_j|^2$ where,

$$c_j(t) = \int \bar{\psi}_j(u - t)x(u)du,$$

and ψ_j is the wavelet centered at $f_c^{(j)}$. This differs from the recipe in (von Tscharner, 2000) but that is mainly because we use here the full complex coefficient. We present here a modification (Borg, 2003) which is based on the Morlet function which in frequency space is given as,

$$\hat{\psi}(f_c, \alpha, f) = \exp\left(-\frac{2\pi^2}{\alpha f_c}(f - f_c)^2\right).$$

For the center frequencies we select, following von Tscharner,

$$f_c^{(j)} = \frac{1}{s}(q + j)^2 \quad (j = 0, \dots, J - 1),$$

determined by the parameters s ("scale") and q . The implementation is given by `EMG_morvt` which is again based on using the `fft` transformation. This function returns a list where `powc` refers to the matrix of the $c^{(j)}[i]$ coefficients, `freqc` to the array with the center frequencies, and `freqm` contains an estimate of the instantaneous mean frequency calculated as the average of the center frequencies weighted with the power coefficients $|c_j|^2$.

4.2.6 Multi resolution analysis, MRA

In the above examples we have relied on the basic libraries that belong to the default setup of the R system. In the next example we will take advantage of a library that provides functions for discrete wavelet analysis. There is for instance a package appropriately named `wavelets` (by Erich Aldrich). In order to install it one enters the command

```
install.packages("wavelets")
```

which will look up a depository and ask you to download the package. When successfully installed it can be loaded by the command

```
library(wavelets)
```

The command `library()` with empty argument will show the packages installed on your system. Information about the package `wavelets` can be obtained by the command

```
help(package = "wavelets")
```

or `??wavelets`. In multi resolution analysis (MRA) we repeatedly apply low- and high-pass filters to a discrete time series which thus can be decomposed into fine and coarse grained parts. The simplest example is the Haar filter. If $x = (x_1, x_2, \dots)$ then Haar low pass and high pass filters produce the series, $a = (a_1, a_2, \dots)$ and If $b = (b_1, b_2, \dots)$, with

$$a_i = \frac{x_{2i} + x_{2i-1}}{\sqrt{2}},$$

$$b_i = \frac{x_{2i} - x_{2i-1}}{\sqrt{2}}.$$

The averaging procedure produces a coarse grained sum version a , while b contains the detail. Symbolically the decomposition can be written $x = (a|b)$. This procedure can be repeated taking a as an input for the decomposition procedure. In this fashion we obtain

$$x = (a^J | b^J | b^{J-1} | \dots | b^1),$$

for a decomposition of order J . The k :th level detail coefficients b^k represent information about the changes in the times series on a time scale proportional to 2^k .

The function `EMGx_mra()` is a wrapper for `mra` in the *wavelet* package. A new feature here is that the function `mra` returns a class object with slots whose names can be accessed by the function `slotNames`. For instance the detail coefficients have the name `D` and the sum coefficients the name `S`. If

```
res <- mra(X)
```

then the vectors with the coefficients are accessed as

```
res@D[[j]], and res@S[[j]],
```

for the level j . The original data can be obtained as a sum of the decomposition,

```
X = res@D[[1]] + res@D[[2]] + ... + res@D[[J]] + res@S[[J]].
```

Thus $\text{res@D}[[j]]$ reflects the signal content on a time scale of the order $2^j \cdot f_s^{-1}$ where f_s is the sampling rate.

4.2.7 Batch processing

As the number of data files grow it is important to be able to process them in one row. This kind of batch processing can be simply implemented in R. We will assume that have a set of data files `name1.asc`, ... , `nameN.asc`. One can collect these paths of these files into `filelist.asc` and write a R-script which opens each of these files for processing. One thing to remember is that the file paths must be on the Unix format using `/` (or `\\`) instead of `\`. The files can also be selected interactively by using the `tk_choose.files()` function,

```
library(tcltk) # load the tcltk package
Filters <- matrix(c("EMG data", ".asc", "All files", "*"),
                 2, 2, byrow = TRUE)

if(interactive()) filelist <- tk_choose.files(filter = Filters)

This will open the "Select files" dialogue and put the selected files into the
filelist variable (with file paths on the Unix format). The following snippet
is a simple example which opens the files in the filelist and plots the first
column to a pdf-file, and writes the standard deviation to a text-file.

# filelist — contains paths to asci files with EMG data
outputpdf <- "C:/EMGanalysis.pdf" # output graphs to this file
outputtxt <- "C:/EMGanalysis.txt" # output text/numbers to this file

pdf(outputpdf) # starts the pdf driver and opens the output pdf-file
fp <- file(outputtxt, "w") # opens text-file for writing
n <- length(filelist)

for(i in 1:n){
  EMG <- read.table(filelist[i], header = FALSE)
  title <- paste("Data from ", filelist[i])
  # this one goes to the text file -->
  cat("Standard deviation = ", sd(EMG$V1),
      " for data EMG$V1 in file ",
      filelist[i], "\n", file = fp)
```

```

    # this one goes to the pdf file -->
    plot(EMG$V1, main = title , type = "l")

}

close(fp) # closes the text file
dev.off() # closes the pdf driver

```

It illustrates how one reads the data, opens a file for writing, where the writing to the file is performed using the function `cat`. (It computes the standard deviation of the time series using the function `sd` and writes it to the file.) This example is easily generalized to more complicated processings.

4.3 Statistics

R is by definition a statistics software package whence all the well-known, and many less well-known, statistical procedures are implemented. Important sub-topics are descriptive statistics, statistical testing, and modeling data. Since our emphasis here is on signal processing we will not go into the statistical methods. At the very basic level we have, for instance, `hist(X)` which computes and plots a histogram of numerical data X , while `plot(ecdf(X))` first calculates the empirical cumulative distribution function (`ecdf`) and the plots the result. The *Student test* is performed by `t.test` and, for instance, `t.test(X, mu = 2)` computes the p -value for the mean to differ from 2, and the 95% confidence interval for the mean of X .

For an introduction to statistical analysis using R we recommend Everitt and Hothorn (2010) which is provided with a R-package `HSAUR2` that contains the codes and the data sets.

5 Conclusions

We have given a brief introduction to some basic features of the R software used as a tool for analyzing and displaying EMG data, and biosignal data in general. Using R it is easy to document the exact procedures employed in analyzing the data so that it can be replicated by other researchers. A next level would be to develop a dedicated REMG package with tools covering various aspects of EMG and related kinesiological data and signals (MMG, ECG, etc). Such a package could be supplied with a representative set of data for testing and demonstrating the analysis methods. Finally we would also like to emphasize the usefulness of R in teaching basic data processing and visualization methods to biomechanics students.

Acknowledgments

The author thanks Maria Finell for gathering the EMG- and balance data used in the examples (supplementary material). He is also indebted to W Jeffrey Armstrong for exchanges on the "intensity analysis" which has resulted in an update of the `EMGfuncs.R` file.

Supplementary material

The file `data.bal` contains quiet standing balance COP-data in ASCII format. First column contains COP X, second column contains COP Y, both in millimeters. The third column contains total vertical force (Newton). The columns are tab-separated (`\t`). Sampling rate is 100 Hz. The file `data.emg` contains the EMG-data in ASCII format. The columns contain data from the muscles Tibialis anterior (right), Lateral Gastrocnemius (right), Medial Gastrocnemius (right), Tibialis anterior (left), Lateral Gastrocnemius (left), Medial Gastrocnemius (left), all sampled at 1000 Hz. The file `emg_analysis.R` is the R-script which demonstrates a few basic analyzing methods with the EMG and the balance data. The script either produces a pdf report (`result.pdf`), or shows the results on the console, depending on the setting of the variable `report` (`TRUE` or `FALSE`). The file `EMGfiles.R` contains a script which demonstrates how to set up batch processing. The file `EMGfuncs.R` contains the basic R-scripts (functions) for analyzing EMG which are employed by `emg_analysis.R`. The files are by default assumed to reside in the archive `C:/EMGR`. The files can be downloaded as <http://terra.chydenius.fi/~frborg/emg/EMGR.zip>, and also from the arXiv site as supplementary material.

References

- Borg F (2003) Filter banks and the "Intensity Analysis" of EMG URL <http://arxiv.org/abs/1005.0696>
- Borg F, Finell M, Hakala I, Herrala M (2007) Analyzing gastrocnemius EMG-activity and sway data from quiet and perturbed standing. *Journal of Electromyography and Kinesiology* 17(5):622–634
- Everitt BS, Hothorn T (2010) A handbook of statistical analyses using R, 2nd edn. CRC Press, Boca Raton
- Hermens HJ, Merletti R, Rix H, Freriks B (1999) SENIAM. The state of the art on

- signal processing methods for surface electromyography. Roessing Research and Development b. v.
- Klemens B (2009) Modeling with data. Tools and techniques for scientific computing. Princeton University Press, Princeton
- Lindström LR, Kadefors R, Petersén I (1977) An electromyographic index for localized muscle fatigue. *Journal of Applied Physiology* 43:750–754
- Press WH, Teukolsky SA, Vetterling WT, Flannery BP (2002) Numerical Recipes in C++. Cambridge University Press, Cambridge
- R Development Core Team (2010) R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, URL <http://www.R-project.org>, ISBN 3-900051-07-0
- Soechting JF, Roberts WJ (1975) Transfer characteristics between EMG activity and muscle tension under isometric conditions in man. *Journal of Physiology* 70:779–793
- von Tscharner V (2000) Intensity analysis in time-frequency space of surface myoelectric signals by wavelets of specified resolution. *Journal of Electromyography and Kinesiology* 10:433–445
- Venable WN, Smith DM, et al (2009) An introduction to R, 2nd edn. Network Theory Ltd, www.network-theory.co.uk/R/intro/
- Vetterli M, Kovačević J (1995) Wavelets and subband coding. Prentice Hall, URL http://waveletsandsubbandcoding.org/Repository/VetterliKovacevic95_Manuscript.pdf
- Willison RG (1963) A method of measuring motor unit activity in human muscle. *Journal of Physiology* 168:35
- Willison RG (1964) Analysis of electrical activity in healthy and dystrophic muscle in man. *Journal of Neurology, Neurosurgery, and Psychiatry* 27:386–394